KEY_Lesson17_Pandas-Subsetting-II

December 10, 2021

1 Subsetting Pandas DataFrames II

In the last lesson, you learned how to subset dataframes by entire rows or entire columns. Now we're going to learn how to do both at the same time!

Let's read in the `tips` dataset again:

[1]:
```
# import the pandas package
import pandas as pd
# set the path
path = 'https://raw.githubusercontent.com/GWC-DCMB/curriculum-notebooks/master/'
# load tips
tips = pd.read_csv(path + 'SampleData/tips.csv')
```

Take a look again at the beginning of the `tips` `DataFrame`:

[2]:
```
# view the beginning of tips
tips.head()
```

[2]:
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

First, let's recall how you would subset three columns, `total_bill`, `day`, and `time`. Let's save it to a variable called `subset1`.

[3]:
```
# subset the colums and save it to subset1
columns = ['total_bill', 'day', 'time']
subset1 = tips[columns]
#subset1 = tips[['total_bill', 'day', 'time']] # alternative method
```

Now, how would you subset the 6th row through the 10th row from the `subset1` dataframe? Let's save it to a variable called `subset2`.

[ ]:
```
# subset the 6th row through the 10th row and save it to tips_subset_rows
subset2 = subset1.iloc[5:11]
```

Now the `subset2` dataframe has just the rows 5 through 10 and three columns. We can even subset rows and columns in the same line of code, instead of doing it on multiple lines like we did above. Let's try combining both `iloc` and square brackets `[]` on one line:

```
[ ]: # subset rows & columns at the same time
     subset3 = tips.iloc[5:11][['total_bill', 'day', 'time']]
```

What do you notice about `subset3`? How does it compare to `subset2`?

```
[ ]: # compare subset2 and subset3
     subset2 == subset3

     # every value printed out is True, so they're exactly the same
```

Now you try! Subset rows 11 and 12 and columns `total_bill` and `tip` on one line of code:

```
[ ]: # subset rows and columns
     tips.iloc[11:13][['total_bill', 'tip']]
```

Sometimes we don't know exactly which row(s) we want to subset ahead of time. What if we want to subset rows that have a certain value in the `time` column? We don't want to scroll through hundreds of rows to find them. The good news is: we don't have to! Let's use the `method` called `query`. Inside the parentheses of `query` we'll enclose a statement in quotes with the name of the column and an expression.

```
[ ]: tips.query('time == "Lunch"')
```

The above cell showed us all the rows where `time` is equal to "Lunch". We had to enclose "Lunch" in quotes above because it's not the name of a column, but a value within the `time` column.

Now you try: subset the rows where the waitress is female and save it to a variable, `female`:

```
[ ]: # subset rows with a female waitress and save it to a variable
     female = tips.query('sex == "Female"')

     # take a look at the beginning
     female.head()
```

Now lets do the same for males. Subset the male waiter data and save it to a variable, `male`:

```
[ ]: # subset the male waiters and save it
     male = tips.query('sex == "Male"')

     # look at the beginning
     male.head()
```

How would you determine the number of male waiters in this `DataFrame`? Think back to the last lesson when we used the `len` function.

```
[ ]: # number of males
     len(male)
```

How about the number of female waitreses?

```
[ ]: # number of females
     len(female)
```

We can use **query** on multiple columns at a time. Let's find out how many tables were served by a female waitress on a Sunday.

```
[ ]: tips.query('sex == "Female" and day == "Sun"')
```

We used the keyword `and` to chain together two statements inside the `query` function. Both statements have to be true for a row to be included.

Besides checking whether values are equal using ==, we can also use greater than, less than, greater than or equal, etc. Try subsetting the rows where the bill is greater than $15 and the tip is less than $2:

```
[ ]: # subset by bill and tip
     tips.query('total_bill > 15 and tip < 2')
```

Instead of `and` we can use the keyword `or` to represent a query where *one* of the two conditions must be fulfilled. Try subsetting where the bill is greater than $15 or the tip is greater than $5:

```
[ ]: # subset by bill or tip
     tips.query('total_bill > 15 or tip > 5')
```

In this lesson, you learned:

- How to use `iloc` and square brackets `[]` at the same time.
- How to use `query` to find rows where the column has a certain value.